A Unified Architecture for Implementation of the Entire Transforms in the H.264/AVC Encoder

Sedighe Ghorbani[1] and Farzad Zargari[2*]

[1] *Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran*
[2]*IT Department of Research Institute for ICT (formerly Iran Telecom Research Center (ITRC)), Tehran, Iran*
s.ghorbani@srbiau.ac.ir, zargari@itrc.ac.ir

Abstract

Integer Discrete Cosine Transform (DCT) is among the techniques used to improve the performance of the H.264/AVC Standard. All the profiles in the H.264/AVC standard support 4×4 integer DCT and the high profiles of this standard support 8×8 integer DCT as well as the 4×4 integer DCT. Various hardware realizations have been proposed for forward and inverse integer DCT in the literature because they are among the computational intensive units in the H.264/AVC standard. In this paper we propose a unified pipelined architecture to realize of the entire forward and inverse DCTs as well as the Hadamard transforms in the H.264/AVC encoder. The synthesis results indicate that our architecture achieves higher clock rate and relatively lower gate count compared to the other published architectures that realize only a number of the transforms in the H.264/AVC encoder.

Keywords: *H.264 encoder, Discreet Cosine Transform (DCT), Integer DCT, Hadmard Transform, Hardware Implementation*

1. Introduction

The H.264/AVC standard [1] achieves remarkable higher compression performance than the previous MPEG and H.26X standards. The higher performance in H.264/AVC is due to various modifications in different coding stages and most of these modifications impose high computational load to the H.264/AVC codec. As a consequence, hardware realization of the computationally intensive parts in the H.264/AVC standard attracted great deal of attention and there are several proposals for the hardware realization of these parts in the literature [2-7]. One of the computationally intensive units in the MPEG and H.26X video coding families is the Discrete Cosine Transform (DCT). Hence, the hardware realization of this unit is even attractive for the pre-H.264/AVC standards and there are proposals for hardware architectures to realize this unit from a long time ago [8, 9] and it is still continuing [10-13].

The H.264/AVC standard employs integer DCT instead of real DCT, which is used in the previous video coding standards. This eliminates any mismatch issue between the encoder and decoder in the inverse transformation [14]. The initial version of H.264/AVC standard supported only 4×4 integer DCT. In order to achieve higher compression performance the amendment called Fidelity Range Extensions (FRExt) was added to the H.264/AVC standard, which adaptively employs both 4×4 and 8×8 transforms in the high profiles [15]. In this way roughly 10% bit-rate reduction can be achieved for various coding parameters [16]. This led

* Corresponding author: Zargari@itrc.ac.ir

to additional complexity of the initial version of the H.264/AVC encoder, which had substantially high computational load. The number of operations for computation of an 8×8 or 4×4 Integer DCT is not very high but since in high profiles theses transforms should be applied to the entire 8×8 or 4×4 blocks in a frame, it will result in a huge computational load and makes the integer discrete cosine transform among main computationally intensive stages in the H.264 encoder [10, 11]. Consequently, the hardware implementation of the integer DCT transform attracted more attention and a number of solutions have been published for hardware implementation of Integer DCT in the H.264/AVC standard [16-19].

In this paper, which is an extended and more detailed version of our previous work [20], we introduce a unified pipelined architecture to realize the entire forward and inverse integer DCTs and Hadamard transforms in the H.264/AVC standard. Since the encoding loop of the H.264/AVC standard requires carrying out all the forward and inverse transforms, the proposed unified architecture is a very powerful accelerator for the H.264/AVC encoder. The proposed architecture is completely in accordance with the reference software of the H.264/AVC standard and the synthesis results indicate that our architecture achieves higher clock rate and has relatively lower hardware cost compared to the previous architectures, which have implemented only a number of the transforms in the H.264/AVC standard.

The rest of the paper is organized as follows. In Section 2 we provide a brief overview of the transforms in the H.264/AVC standard and discuss their existing hardware implementations. The proposed architecture for implementation of the entire transforms in the H.264/AVC standard is explained in Section 3. The synthesis results for the given architecture and comparison with the other exiting implementations are presented in Section 4 followed by concluding remarks given in Section 5.

2. Background

In the H.264/AVC standard the forward and the inverse integer DCT are defined respectively in (1) and (2) as:

$$Y=AXA^T \quad => \quad Y= C_fXC_f^T \otimes E_f \tag{1}$$

$$X=A^TYA \quad => \quad X=C_i^T(Y\otimes E_i)C_i \tag{2}$$

The $C_fXC_f^T$ and $C_i^TWC_i$ parts in the above equations are called ‘core’ transforms [21]. ‘Core’ transform is a two dimensional transform, which can be decomposed into two one dimensional transforms. The first one dimensional transform is applied to the rows of the input pixels and the second one dimensional transform is applied to the columns of the one dimensional transform coefficients of the first stage (Figure 1).

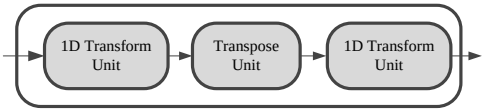

Figure 1. Decomposing 2D Integer DCT into two 1D Integer DCT

The C_f and C_i matrices given in (3) indicate the ‘core’ transform matrix of the forward and inverse 4×4 integer DCT in the H.264/AVC standard, respectively.

$$C_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \quad C_i = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \tag{3}$$

Figure 2 shows a fast hardware realization for 4×4 forward integer transform using adders and shifters and Figure 3 indicates a fast hardware realization for 4×4 inverse integer transform, both given in [14].

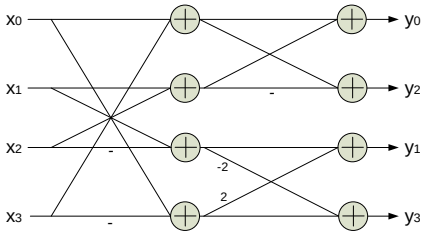

Figure 2. Fast Realization for Forward 4×4 Integer DCT

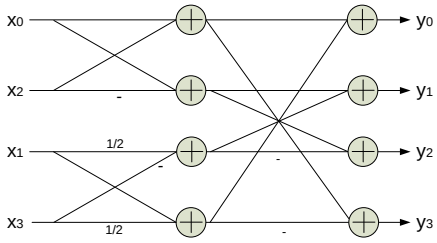

Figure 3. Fast Realization for Inverse 4×4 Integer DCT

The Hadamard transform is another 2D transform which is used in the H.264/AVC standard and its 'core' transform matrix is:

$$H_{4\times4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{4}$$

Authors in [22] employed the butterfly architecture for fast hardware implementation of the Hadamard transform (Figure 4).

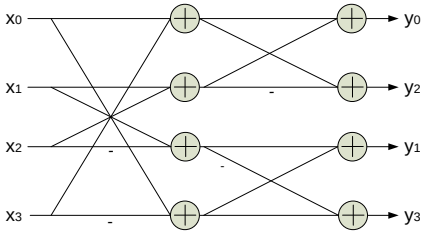

Figure 4. Fast Realization for 4×4 Hadamard Transform

Since $H^T_{4\times4} = H_{4\times4}$, the hardware implementation given in Figure 4 can be used for both forward and inverse Hadamard transforms. The H.264/AVC standard uses 2×2 Hadamard transform as well. The 'core' transform matrix for 2×2 Hadamard transform is as:

$$H_{2\times2} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{5}$$

Since the hardware implementation of 4×4 Hadamard transform can also be employed for 2×2 Hadamard transform, a dedicated hardware implementation for 2×2 Hadamard transform

is not required. The authors in [22] unified all the aforementioned fast schemes and introduced a unified circuit for realization of the entire 4×4 and 2×2 transforms in H.264/AVC (Figure 5).

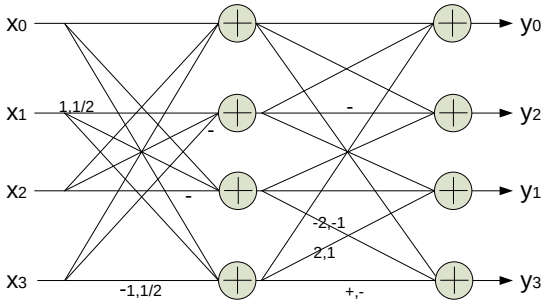

Figure 5. Fast Multipurpose Architecture for all 4×4 Transforms in H.264/AVC

The FRExt of H.264/AVC standard uses both 4×4 and 8×8 integer DCT transforms adaptively for high resolution video applications. The 'core' transform matrix for 8×8 integer DCT is:

$$C=\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \frac{3}{2} & \frac{5}{4} & \frac{3}{4} & \frac{3}{8} & -\frac{3}{8} & -\frac{3}{4} & -\frac{5}{4} & -\frac{3}{2} \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 & -1 & -\frac{1}{2} & \frac{1}{2} & 1 \\ \frac{5}{4} & -\frac{3}{8} & -\frac{3}{2} & -\frac{3}{4} & \frac{3}{4} & \frac{3}{2} & \frac{3}{8} & -\frac{5}{4} \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ \frac{3}{4} & -\frac{3}{2} & \frac{3}{8} & \frac{3}{4} & -\frac{3}{4} & -\frac{3}{8} & \frac{3}{2} & -\frac{3}{4} \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} & -\frac{1}{2} & 1 & -1 & \frac{1}{2} \\ \frac{3}{8} & -\frac{3}{4} & \frac{5}{4} & -\frac{3}{2} & \frac{3}{2} & -\frac{5}{4} & \frac{3}{4} & -\frac{3}{8} \end{bmatrix} \quad (6)$$

Authors in [17] have given an architecture for implementing the forward 8×8 integer DCT based on the 8×8 DCT algorithm at the H.264/AVC reference software. The architecture in [17] uses five stages of adders, which either reduce the achievable highest frequency in non-pipelined realization or increase the number of pipeline stages in the pipelined realization. In [18] a hardware implementation has been introduced for the forward 8×8 integer DCT. It requires all the 8×8 elements of the block simultaneously and as a negative result, it also needs high amount of hardware resources. A flexible architecture is given in [19] for realizing all inverse transforms in H.264/AVC standard, but the proposed architecture does not support forward transforms.

The architecture in [23] unifies 2D 4×4 and 2×2 with 8×8 1D transforms based on matrix manipulations but the resulted unified architecture is not compliant with the H.264/AVC reference because as we will show in the next section it is very important to consider the way that reference software has been used to implement matrix multiplications. Moreover the proposed method in [23] suffers from high number of processing elements including 44 adders or subtractors. In this paper we introduce a unified architecture for the implementation of the entire transforms in H.264/AVC standard which is completely compliant with the H.264/AVC reference software. The proposed architecture requires only 32 adders or

subtractors and synthesis results indicate that it needs lower area compared to the other reported synthesized architecture that implement only a number of the transforms in the H.264/AVC standard. Meanwhile, it achieves higher maximum frequency and throughput compared to the existing architectures.

In the following section we introduce a flexible architecture for implementation of forward and inverse 8×8 integer DCT of the H.264/AVC standard. The proposed flexible architecture is then expanded to realize the 4×4 and 2×2 transforms in the H.264/AVC standard too.

3. Proposed Architecture

Using the proposed method in [24] multiplication by the 8×8 integer DCT matrix given in (6), can be decomposed into multiplication by two 4×4 matrices as given in (7).

$$A_{f\_1}=\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix} \qquad A_{f\_2}=\begin{bmatrix} \frac{3}{2} & \frac{5}{4} & \frac{3}{4} & \frac{3}{8} \\ \frac{5}{4} & -\frac{3}{8} & -\frac{3}{2} & -\frac{3}{4} \\ \frac{3}{4} & -\frac{3}{2} & \frac{3}{8} & \frac{5}{4} \\ \frac{3}{8} & -\frac{3}{4} & \frac{5}{4} & -\frac{3}{2} \end{bmatrix} \tag{7}$$

$A_{f\text{-}1}$ and $A_{f\text{-}2}$ can be used to produce the 1D 8×8 integer DCT coefficients as given in (8):

$$\begin{bmatrix} y(0) \\ y(2) \\ y(4) \\ y(6) \end{bmatrix}=\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}\begin{bmatrix} x(0)+x(7) \\ x(1)+x(6) \\ x(2)+x(5) \\ x(3)+x(4) \end{bmatrix} \tag{8-a}$$

$$\begin{bmatrix} y(1) \\ y(3) \\ y(5) \\ y(7) \end{bmatrix}=\begin{bmatrix} \frac{3}{2} & \frac{5}{4} & \frac{3}{4} & \frac{3}{8} \\ \frac{5}{4} & -\frac{3}{8} & -\frac{3}{2} & -\frac{3}{4} \\ \frac{3}{4} & -\frac{3}{2} & \frac{3}{8} & \frac{5}{4} \\ \frac{3}{8} & -\frac{3}{4} & \frac{5}{4} & -\frac{3}{2} \end{bmatrix}\begin{bmatrix} x(0)-x(7) \\ x(1)-x(6) \\ x(2)-x(5) \\ x(3)-x(4) \end{bmatrix} \tag{8-b}$$

Multiplication by $A_{f\text{-}1}$ can be further decomposed to multiplication by two 2×2 matrices as:

$$\begin{bmatrix} y_{(0)} \\ y_{(4)} \end{bmatrix}=\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} x_0+x_3 \\ x_1+x_2 \end{bmatrix} \qquad \begin{bmatrix} y_{(2)} \\ y_{(6)} \end{bmatrix}=\begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & -1 \end{bmatrix}\begin{bmatrix} x_0-x_3 \\ x_1-x_2 \end{bmatrix} \tag{9}$$

Considering the decomposition for $A_{f\text{-}1}$ given in (9), we propose to employ the butterfly architecture of Figure 6 to implement the multiplication by $A_{f\text{-}1}$. In order to compute the second matrix multiplication of the 1D forward 8×8 Integer DCT transform, we introduce the architecture of Figure 7 to realize multiplication by $A_{f\text{-}2}$.

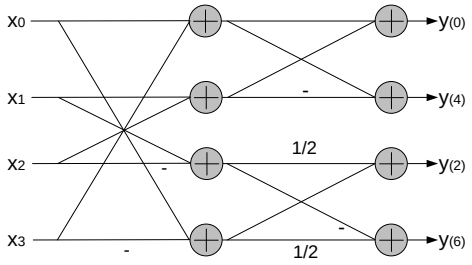

Figure 6. Fast Realization for Multiplication by A_{f-1}

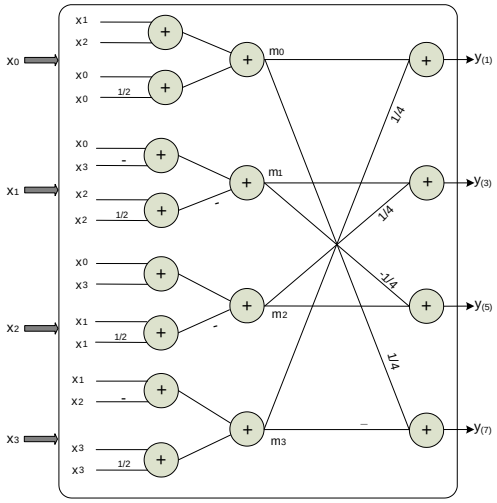

Figure 7. The Proposed Architecture for Multiplication by A_{f-2}

The 8×8 inverse integer DCT matrix in the H.264/AVC standard can be decomposed to multiplication by two 4×4 matrices as given in (10).

$$A_{i\_1} = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \qquad A_{i\_2} = \begin{bmatrix} \frac{3}{2} & \frac{5}{4} & \frac{3}{4} & \frac{3}{8} \\ \frac{5}{4} & -\frac{3}{8} & -\frac{3}{2} & -\frac{3}{4} \\ \frac{3}{4} & -\frac{3}{2} & \frac{3}{8} & \frac{5}{4} \\ \frac{3}{8} & -\frac{3}{4} & \frac{5}{4} & -\frac{3}{2} \end{bmatrix} \tag{10}$$

A_{i-1} and A_{i-2} can be used to perform inverse 8×8 integer DCT transform as:

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} y(0) \\ y(2) \\ y(4) \\ y(6) \end{bmatrix} + \begin{bmatrix} \frac{3}{2} & \frac{5}{4} & \frac{3}{4} & \frac{3}{8} \\ \frac{5}{4} & -\frac{3}{8} & -\frac{3}{2} & -\frac{3}{4} \\ \frac{3}{4} & -\frac{3}{2} & \frac{3}{8} & \frac{5}{4} \\ \frac{3}{8} & -\frac{3}{4} & \frac{5}{4} & -\frac{3}{2} \end{bmatrix} \begin{bmatrix} y(1) \\ y(3) \\ y(5) \\ y(7) \end{bmatrix} \tag{11.a}$$

$$\begin{bmatrix} x(7) \\ x(6) \\ x(5) \\ x(4) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} y(0) \\ y(2) \\ y(4) \\ y(6) \end{bmatrix} - \begin{bmatrix} \frac{3}{2} & \frac{5}{4} & \frac{3}{4} & \frac{3}{8} \\ \frac{5}{4} & -\frac{3}{8} & -\frac{3}{2} & -\frac{3}{4} \\ \frac{3}{4} & -\frac{3}{2} & \frac{3}{8} & \frac{5}{4} \\ \frac{3}{8} & -\frac{3}{4} & \frac{5}{4} & -\frac{3}{2} \end{bmatrix} \begin{bmatrix} y(1) \\ y(3) \\ y(5) \\ y(7) \end{bmatrix} \tag{11.b}$$

The matrix A_{i-1} is identical with 4×4 inverse integer DCT matrix C_i. Hence, the scheme in Figure 3 can be used to implement A_{i-1}. We have proposed in Figure 5 and Figure 6 to merge the schemes to give the combined architecture of Figure 8 to implement the entire 4×4 transforms in the H.264/AVC standard and also the transforms by matrices A_{f-1} and A_{i-1}.

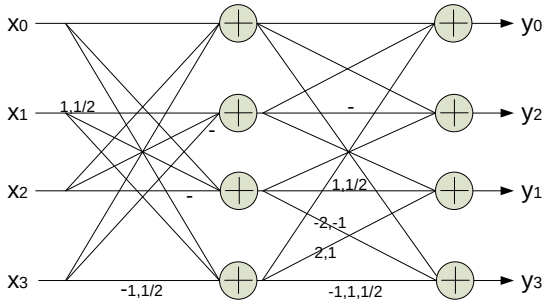

Figure 8. Fast Realization for Multiplication by Af-1 ,Ai-1 and all 4×4 Transform Matrices in H.264/AVC

Considering (5) we infer that the output of the adders in the first stage of Figure 8 can be used to implement 2×2 Hadamard transform. Hence, we propose the architecture in Figure 9, referred to as Ext_1D_Transform hereafter, to implement multiplication by matrices A_{f-1} and A_{i-1} and the entire 4×4 and 2× 2 transforms in the H.264/AVC standard. The total computational complexity in Figure 8 is 3 adders, 3 subtractors, 2 adder/subtractors and 6 shifters.

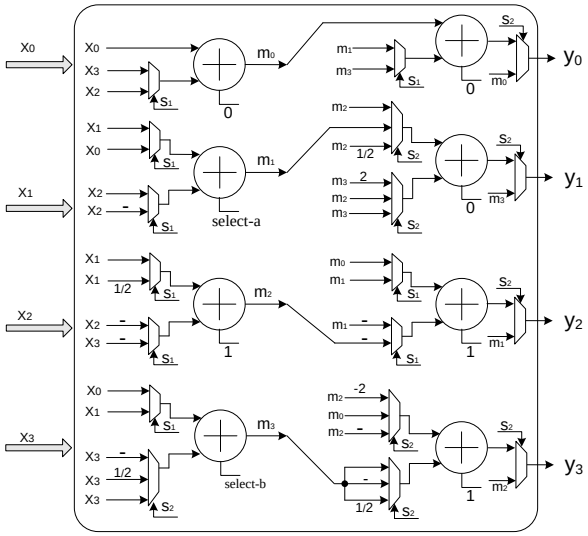

Figure 9. Fast Realization for Multiplication by A_{f-1} ,A_{i-1} and all 4×4 and 2×2 Transforms in H.264/AVC (Ext_1D_Transform)

In order to perform 8×8 inverse integer DCT transform, realization of multiplication by A_{i-2} is necessary. Even though A_{i-2} is identical with A_{f-2}, the way its multiplication is implemented in the H.264/AVC reference software differs from that of A_{f-2}. As an example to compute the first element of output vector resulted from multiplication by A_{f-2}, the reference software first computes two intermediate variables $m_{0(f)}$ and $m_{3(f)}$ as:

$$m_{0(f)} = x_1 + x_2 + ((x_0 >> 1) + x_0) \quad (12\text{-a})$$

$$m_{3(f)} = x_1 - x_2 + ((x_3 >> 1) + x_3) \quad (12\text{-b})$$

and then calculates the first element of the vector resulted from multiplication by A_{f-2}:

$$y_{0(f)} = m_{0(f)} + (m_{3(f)} >> 2) \quad (13)$$

On the other hand to perform the inverse transform, the reference software generates $m_{0(i)}$ and $m_{3(i)}$ intermediate variables:

$$m_{0(i)} = x_1 + x_2 + x_0 + (x_0 >> 1) \quad (14\text{-a})$$

$$m_{3(i)} = -x_1 + x_2 - x_3 - (x_3 >> 1) \quad (14\text{-b})$$

and the first element of multiplication by A_{i-2} is computed as:

$$y_{0(i)} = m_{0(i)} - (m_{3(i)} >> 2) \quad (15)$$

It is worth noting that arithmetic right shift results in different rounding effects on positive and negative numbers e.g. positive numbers approach zero by arithmetic right shift, while the two's complement representation of negative numbers approach -1. Due to this fact the results for $y_{0(f)}$ and $y_{0(i)}$ may be different, though they are realizing the same matrix multiplication. As an example for the input matrix $(x0, x1, x2, x3)^T = (0, 1, -2, 0)^T$ the first element of the output matrix resulted from multiplication by A_{f-2} and A_{i-2} will be: $y_{0(f)} = -1 \neq y_{0(i)} = 0$. In fact performing matrix multiplications by identical matrices does not guarantee compliance with the H.264/AVC standard. It means that the methods such as [23] which only consider matrix multiplication by using arbitrary mathematical manipulations will not guarantee compliance with the H.264/AVC standard. Hence, we should use different architectures to carry out multiplication by matrices A_{i-2} and A_{f-2} in order to keep consistency with the H.264/AVC reference software. We merged the two different architectures for implementation of multiplication by matrices A_{i-2} and A_{f-2} in the forward and inverse 8×8 integer DCT to propose the unified architecture in Figure10. The architecture performs multiplication by matrices A_{f-2} or A_{i-2} when the selection signal 'S' is 0 or 1, respectively.

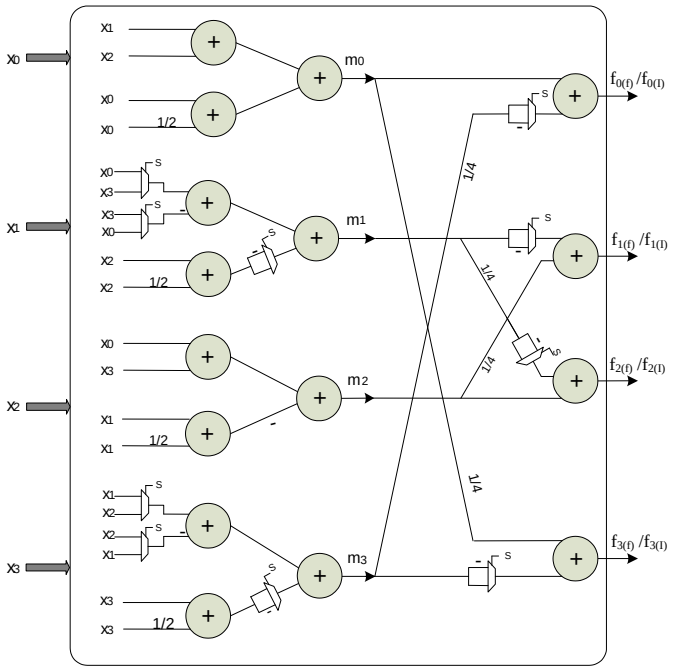

Figure 10. The Proposed Architecture for Multiplication by Matrices A_{i-2} and A_{f-2} in Forward and Inverse 8×8 Integer DCT

Since we also want to use the architecture in Figure 10 to perform the 4×4 and 2×2 transforms of the H.264/AVC standard, we proposed the architecture in Figure 11. The architecture in Figure 11, referred to as the New_1D_transform hereafter, can realize multiplication by matrices A_{i-2} *and* A_{f-2} and the entire multiplications required for the 4×4 and 2×2 transforms. The total computational complexity of the architecture in Figure 11 is 7 adders, 3 subtractors, 6 adder/subtractors and 12 shifters.

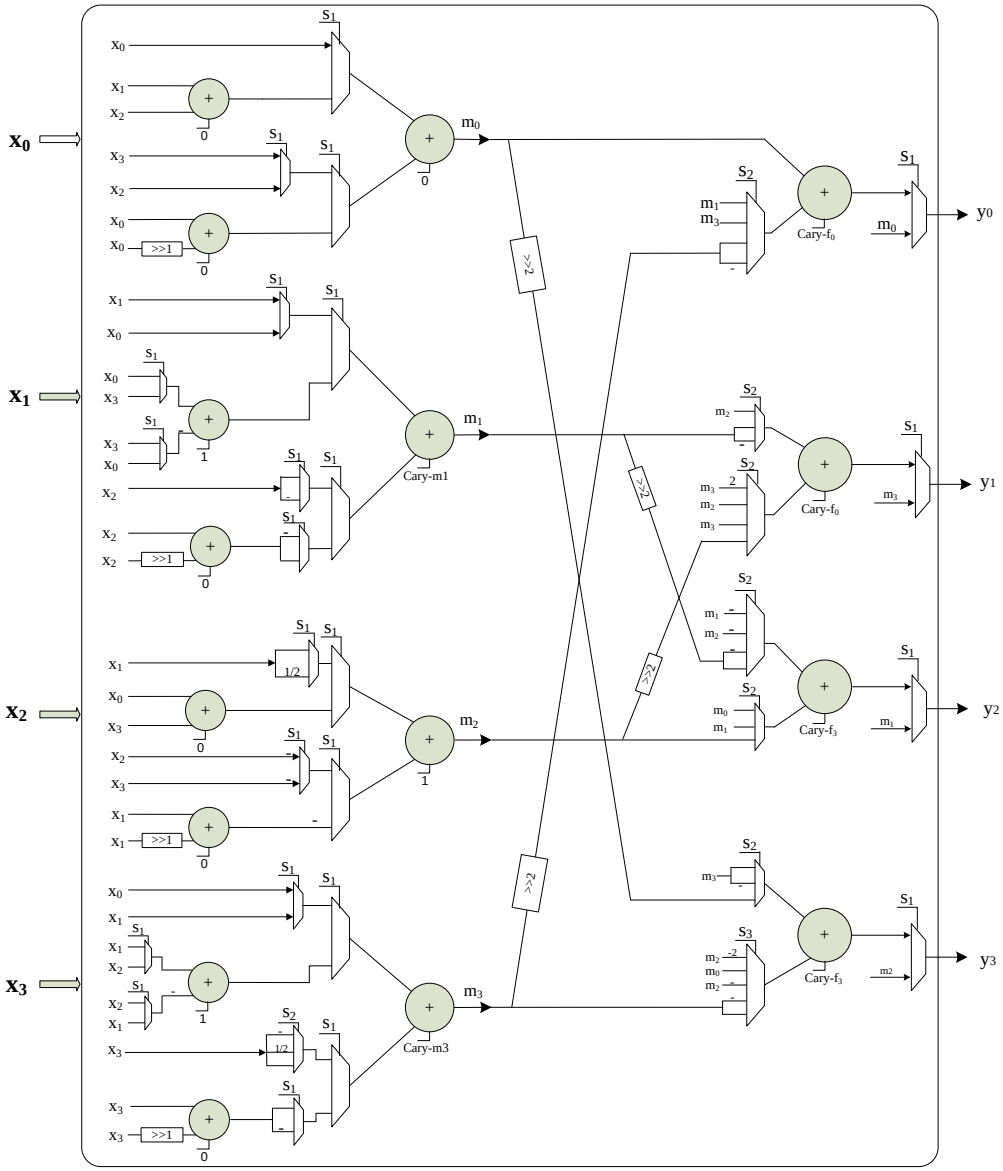

Figure 11. The Proposed Architecture for Realizing Multiplication by the Matrices A_{i-2} , A_{f-2} and all the 4×4 and 2×2 Transforms in H.264/AVC (New_1D_transform)

We used the Ext_1D_Transform along with the New_1D_transform to derive a unified architecture for the realization of 1D transform part of the entire 2D transforms in the H.264/AVC standard (Figure 12). As (8) and (11) indicate, besides multiplication by A_{f-1}, A_{f-2}, A_{i-1} and A_{i-2} an extra stage of adders is required in the input and output of the matrix multipliers for the forward and inverse 8×8 transforms, respectively. The adders in stage 1 of Figure 12 perform the additions in (8), while the adders in stage 5 of Figure 12 carry out the additions in (11).

It is worth noting that when the NEW_1D_Transform is used to perform 8×8 transforms it has one more pipeline stage than the Ext_1d_Transform. Hence, in order to use these two units in the same pipelined architecture, we added one more register stage to the Ext_1D_Transform at the architecture in Figure 12. The added registers are used only during 8×8 transforms and are bypassed otherwise.

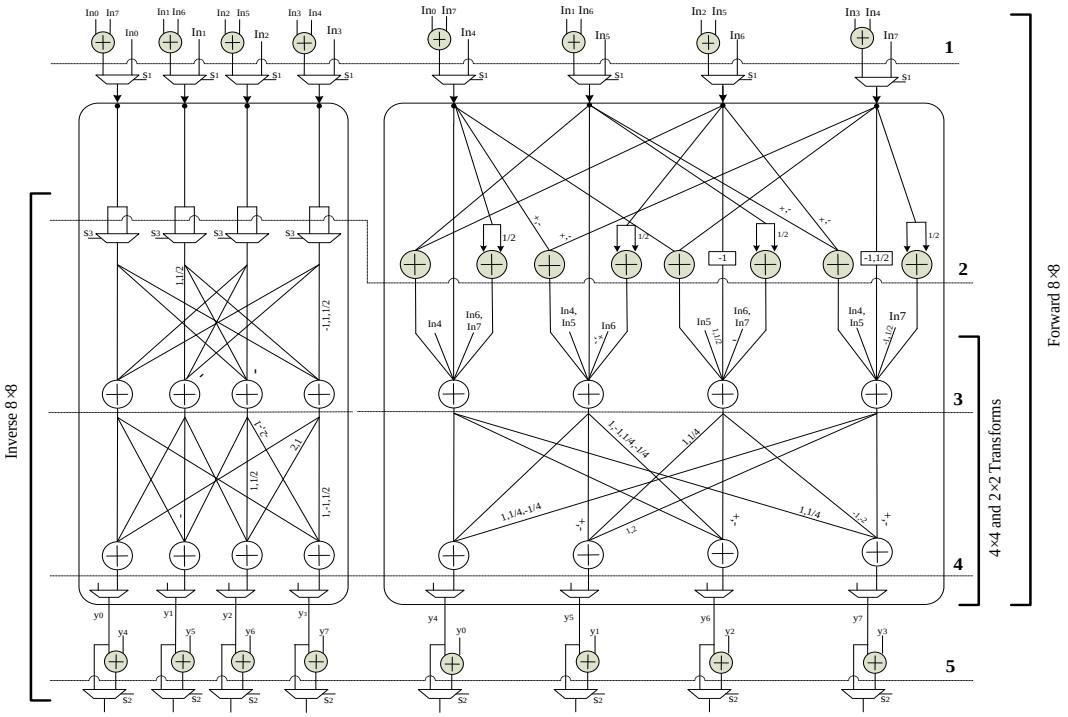

Figure 12. 1D Transform Architecture for all Transforms in H.264/AVC

Since the eight adders in stage 1 of Figure 12 are used only in the forward 8×8 integer DCT and the eight adders in stage 5 are applied only in inverse 8×8 integer DCT, we employed an array of eight adders for both stages and switch them between first and last stages in forward and inverse 8×8 transforms, respectively. Figure 13 indicates the final unified 1D transform architecture for the entire transforms in H.264/AVC.

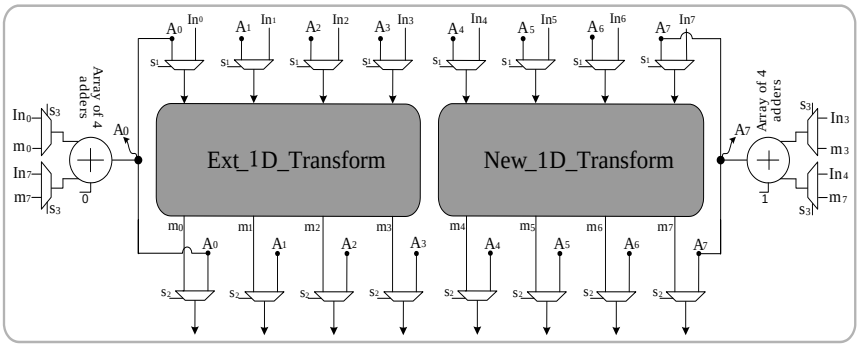

Figure 13. Final unified 1D transform architecture for all transforms in H.264/AVC

Two 1D transform modules of Figure 13 are used along with the transposing architecture are used to give architecture for implementation of the entire 2D transforms in the H.264/AVC standard (Figure 14). The total computational complexities in each 1D part of Figure 15 are 14 adders, 10 subtractors, 8 adder/subtractors and 18 shifters. When the pipeline is full, the proposed architecture requires eight clock cycles to perform an 8×8 transform or four clock cycles to perform two 4×4 or eight 2×2 transforms.

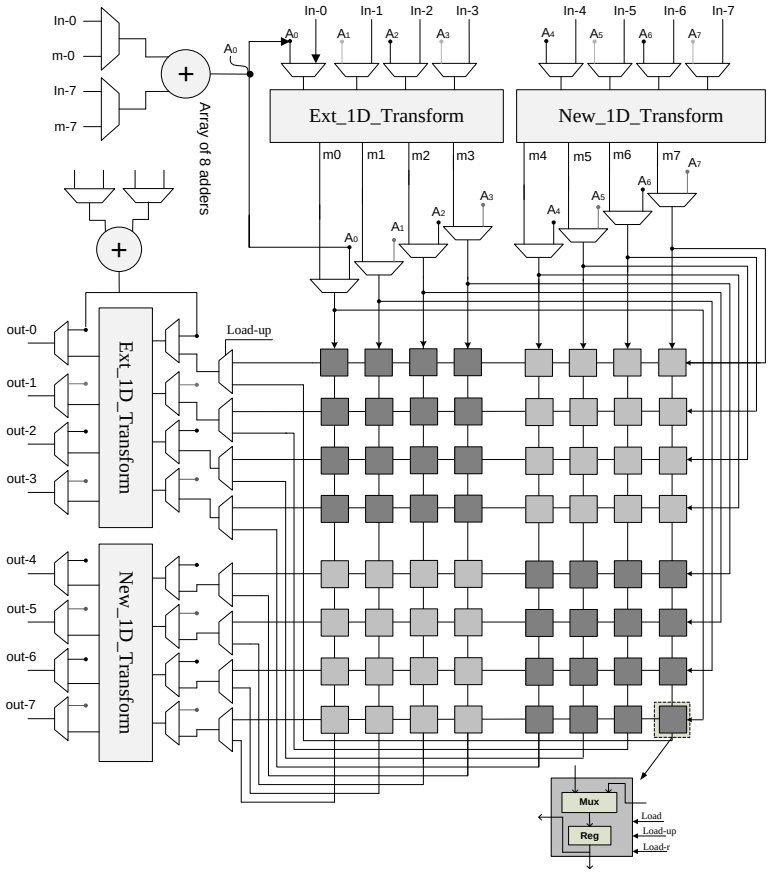

Figure 14. The Proposed Architecture for Computation of all 2D Transforms in H.264/AVC Standard

4. Synthesis Results and Comparison

We used VHDL to describe the proposed pipelined architecture and it was functionally tested by comparing the outputs with those of H.264/AVC reference software in the coding of various image sequences. The TSMC 0.18 μm and 0.09 μm standard cell libraries are used to synthesize our hardware. Furthermore, we employed the Prime Power^{TM} EDA tool to estimate the dynamic and static power consumption. Table 1 lists synthesis and power analyses results.

Table 1. Synthesis Results for the Proposed Architecture

Technology	Gate count	Critical path (ns)	Power(mW)
0.18 μm	*33,000*	*6.46*	*161.672@ 125 MHz*
0.09 μm	*41,445*	*1.41*	*23.727@ 500 MHz*

Table 1 indicates that the pipelined architecture can achieve to a maximum speed of higher than 150 MHz or700MHz using 0.18 μm or 0.09 μm libraries, respectively.

Table 2 lists the synthesis results using 0.18 μm library for our proposed architecture and a number of other fast architectures for realizing a number of the 2D transforms in the H.264/AVC standard. The synthesis results given in Table 2 indicate that our unified architecture with carry ripple adders achieves higher maximum clock rate, higher throughput and lower gate count compared to the previous fast architectures that implemented a number of 2D transforms in the H.264/AVC standard. It is worth noting that, even though the authors in [19] reported higher maximum frequency, each stage of the pipeline architecture in [19] includes two stages of adders while in our pipelined architecture requires only one stage of adders. It means that the higher maximum speed in [19] is not due to its architecture but it is because of employing faster implementation for adders. To give an example for employing fast adders in the proposed architecture, we synthesized the proposed architecture using CLAs and the simulation results given in Table 2 indicate about 30% increase in the maximum frequency compared to the carry ripple adder implementation.

Table 2. Synthesis Results for Different Architectures (using 0.18 micron library)

Ref	function	Gate count	Max. Speed (MHz)	Power (mW)	Pixels/ cycle
[22] redesigned by [25]	*f & inv 4,Had 4*	*6274*	*100*	*N.A.*	4
[25]	*f & inv 4,Had 4*	*6482*	*100*	*N.A.*	8
[19]	*inv 8,4,2*	*18500*	*125*	*N.A.*	8
[26]	*inv 4,8 AVC and inv 8 AVS ,Had 2,4*	*34335*	*100*	*34.266@62.5MHz*	*2,4,8 at 2×2, 4×4,8×8mode*
ours	*f & inv 4,8 ,Had 4,2*	*31263*	*118*	*78@62.5MHz*	*8*
ours (using CLAs)	*f & inv 4,8 ,Had 4,2*	*33000*	*154*	*82.748@62.5MHz*	*8*

5. Conclusion

In this paper a unified architecture with minimum redundancy for realization of 2D transforms in H.264/AVC is proposed. It exploits the similarities among the fast architectures for 4×4 and 2×2 integer DCT matrices and decomposition matrices of the 8× 8 forward and

inverse integer DCT. The devised unified architecture complies with the H.264/AVC reference software. Synthesis results indicate that our architecture, which realizes the entire transforms in the H.264/AVC standard, can process higher number of pixels per clock at higher clock frequency compared to the other published architectures, which implement a number of the transforms in H.264/AVC. Moreover, by comparing the gate count of our architecture with others we conclude that it requires relatively smaller cheap area than that of the individual and separate realizations. Hence, the unified architecture is a fast and resource efficient implementation for the entire transforms in the H.264/AVC standard and can be used in high speed real time H.264/AVC encoding applications.

References

[1] ITU-T Rec. H.264 / ISO/IEC 11496-10, "Advanced video coding for generic audiovisual services", **(2005)** March.
[2] L. Li, Y. Song, S. Li, T. Ikenaga and S. Goto, "A Hardware Architecture of CABAC Encoding and Decoding with Dynamic Pipeline for H.264/AVC", Journal of Signal Processing Systems, vol. 50, no. 1, **(2008)**, pp. 81-95.
[3] Y. H. Chen, T. C. Chen, S. Y. Chien, Y. W. Huang and L. G. Chen, "VLSI Architecture Design of Fractional Motion Estimation for H.264/AVC", Journal of Signal Processing Systems, vol. 53, no. 3, **(2008)**, pp. 335-347.
[4] C. L. Hsu and Y. S. Huang, "A Fast-Deblocking Boundary-strength Based Architecture Design of Deblocking Filter in H.264/AVC Applications", Journal of Signal Processing Systems, vol. 52, no. 3, **(2008)**, pp. 211-229.
[5] M. Kthiri, H. N. Loukil, A. Atitallah, P. Kadionik and D. Dallet, *et al.*, "FPGA architecture of the LDPS Motion Estimation for H.264/AVC Video Coding", Journal of Signal Processing Systems, Online First™, **(2011)** August 10.
[6] G. A. Ruiz and J. A. Michell, "An Efficient VLSI Architecture of Fractional Motion Estimation in H.264 for HDTV", Journal of Signal Processing Systems, vol. 62, no. 3, **(2011)**, pp. 443-457.
[7] M. S. Sayed, W. Badawy and G. Jullien, "Interpolation-Free Fractional-Pixel Motion Estimation Algorithms with Efficient Hardware Implementation", Journal of Signal Processing Systems, Online First™, 5 October **(2010)**.
[8] W. Liebsch, "Parallel architecture for VLSI implementation of a 2-dimensional discrete cosine transform for image coding", Third International Conference on Image Processing and its Applications, **(1989)**, pp. 609–612.
[9] W. K. Cham, C. S. O. Choy and W. K. Lam, "A 2-D integer cosine transform chip set and its applications", IEEE Trans. Consum. Electron., vol. 38, no.2, **(1992)** May, pp. 43–47.
[10] R. A. Arce-Nazario and D. Rodr ́guez, "Mapping of Discrete Cosine Transforms onto Distributed Hardware Architectures", Journal of Signal Processing Systems, vol. 53, no. 3, **(2008)**, pp. 367-382.
[11] J. Park and K. Roy, "A Low Complexity Reconfigurable DCT Architecture to Trade off Image Quality for Power Consumption, Journal of Signal Processing Systems, vol. 53, no. 3, **(2008)**, , pp. 399-410.
[12] S. C. Wang, "Recursive algorithm, architectures and FPGA implementation of the two-dimensional discrete cosine transform", IET Image Process., vol. 2, no. 6, **(2008)**, pp. 286–294.
[13] Z. Wu, J. Sha, Z. Wang, L. Li and M. Gao, "An Improved Scaled DCT Architecture", IEEE Transactions on Consumer Electronics, vol. 55, no.2, **(2009)** May, pp. 685-689.
[14] H. Malvar, A. Hallapuro, M. Karczewicz and L. Kerofsky "Low-complexity transform and quantization in H.264/AVC", IEEE Trans. Circuit syst. Video Techno, vol. 3, no. 7, **(2003)** July, pp. 598-603.
[15] G. Sullivan P. Topiwala and A. Luthra, "The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions", Proc. SPIE Conference on Applications of Digital Image Processing XXVII, Special Session on Advances in the New Emerging Standard: H.264/AVC I, Denver, CO, **(2004)** August, pp. 454–474.
[16] J. S. Park and T. Ogunfunmi "A New Hardware Implementation of The H.264 8×8 Transform and Quantization", Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, **(2009)**, pp. 585-588.
[17] R. Korah and J. R. P. Perinbam, "FPGA Implementation of Integer Transform and Quantizer for H.264 Encoder", Journal of Signal Processing Systems, vol. 53, no. 3, **(2008)**, pp. 261-269.

[18] I. Amer, W. Badawy and G. Jullien, "A High Performance Hardware Implementation of the H.264 Simplified 8x8 Transformation and Quantization", Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, vol.2, **(2005)** March, pp. 1137-1140.
[19] Y. C. Chao, H. H. Tsai, Y. H. Lin, J. F. Yang and B. D. Liu, "A Novel Design for Computation of all Transforms in H.264/AVC Decoders", Proc. of IEEE International Conference on Multimedia and Expo, **(2007)** July, pp. 1914-1917.
[20] F. Zargari and S. Ghorbani, "A Hardware Sharing Architecture for Implementing the entire Transforms in H.264/AVC Video Coding Standard", 15th IEEE International Symposium on Consumer Electronics (ISCE2011), **(2011)** June, pp. 14-17.
[21] I. E. G. Richardson, "H.264 and MPEG-4 Video Compression: Video Coding for Next-generation", Wiley, **(2003)**.
[22] T. C. Wang, Y. W. Huang, H. C. fang and L. G. Chen, "Parallel 4×4 2_D Transform and Inverse Transform Architecture for Mpeg-4 AVC/H.264", Proc. of IEEE int. Symp. on Circuits and System (ISCAS'03), **(2003)**, pp. 800-803.
[23] P. -H. Chen, H. -M. Chen, M. -C. Shie, J. -C. Chen and J. -M. Chang, "An Unified Architecture of All Transforms for H.264/AVC Codec", in Proc. IEEE 3CA, **(2010)** May, pp. 476-479.
[24] A. Madisetti and A. N. Willson, "A 100 MHz 2-D 8×8 DCT/IDCT processor for HDTV applications", IEEE Trans. Circuits Syst. Video Technol., vol. 5, no. 2, **(1995)** April, pp. 158–165.
[25] K. H. Chen, J. L. Guo and J. S. Wang "A High-Performance Direct 2-Dtransform Coding IP Design for MPEG-4 AVC/H.264", IEEE Transaction on circuits and system for video technology, vol. 16, no. 4, **(2006)** April, pp. 472-483.
[26] G. A. Su and C. P. Fan "Low-Cost Hardware-Sharing Architecture of Fast 1-D Inverse Transforms for H.264/AVC and AVS Applications", IEEE Transaction on Circuits and system-II, vol. 55, no. 12, **(2008)** December, pp. 1249-1253.

Authors

Sedighe Ghorbani

She received her B.Sc. degree in computer engineering from Iran Shahed University, Tehran, Iran. She is currently M.Sc. student in the computer engineering department of Iran Islamic Azad University, Science and Research Campus, Tehran, Iran. Her research interests include hardware implementation of image and video coding standards.

Farzad Zargari

He received his B.Sc. degree in Electrical Engineering from Sharif University of Technology and his M.Sc. and Ph.D. degrees in Electrical Engineering from University of Tehran, all in Tehran, Iran.

He is currently a research associate at the information technology department of research institute for ICT, formerly known as Iran Telecom Research Center (ITRC), Ministry of Telecommunications and Information Technology of Iran. He is also a teaching academic staff in the computer engineering department of Science and Research branch of Islamic Azad University. His research interests include multimedia systems, image and video signal processing algorithms, and hardware implementation of image and video coding standards.